Comparison of Different Machine Learning Models on High Dimensional Gene Expression Data.

DATASCI 223

By Soha Shahidi and Zi Li

Objective

The objective of this project is to use machine learning algorithms to predict survival outcomes of patients with kidney renal clear cell carcinoma (KIRC) using gene expression data.

Data

The dataset used came from The Cancer Genome Atlas (TCGA), and contains records from 243 subjects with kidney renal clear cell carcinoma. The dataset in its original form contains information on each subject's age, gender, tumor grade, cancer stage, survival status, messenger RNA expression, micro RNA expression, and copy number variation. There are 16382 predictors in total. The dataset is tabular in nature.

We aimed to create a model that would use gene expression data along with base demographics (age and gender) to predict survival outcomes for patients. To prepare the data, we removed cancer grade and tumor grade as predictors, as those are also potential outcomes that could have too much influence over survival and thus diminish our main goal of using gene expression data as the main predictor. We also removed the "Feature" column as it contained patient identification numbers, which would be useless as predictors. In the end, the dimensions of the dataset were 243 rows by 16379 columns. With a dataset like this, the ideal model would be good at handling very high dimensional data, especially since there are not many rows to rely on.

Each model was trained using 75% of the data, with 25% being withheld for testing.

Models

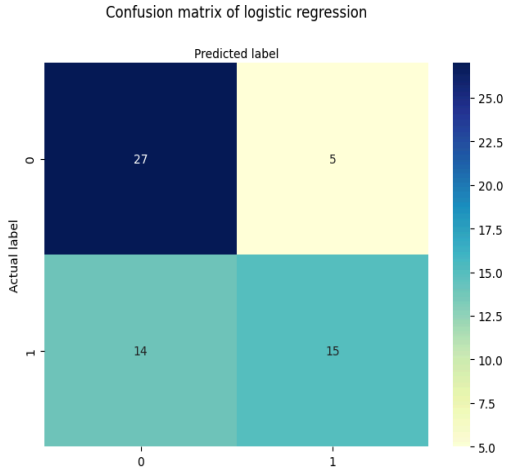
Logistic Regression

By Zi Li

Dependencies: pandas, numpy, train_test_split, LogisticRegression, metrics
Methods: I first removed insignificant features such as feature (patient ID), grade of tumor (identical to the outcome variable) and stage of cancer (identical to the outcome variable) to simplify the model. I chose survival status as the outcome variable. The primary advantage of using logistic regression is that it is more computationally efficient than more advanced machine learning algorithms. I used train_test_split

to split the data into 75% training set and 25% testing set. LogisticRegression was used to train the logistic model and to predict the survival status on the testing set. Finally, accuracy and confusion matrix were shown to see how well the predictive performance in each subgroup of the outcome variable.
Results: The model has an accuracy of 68.9%.

Figure 1. *Confusion matrix (0 = Alive, 1 = Died):*



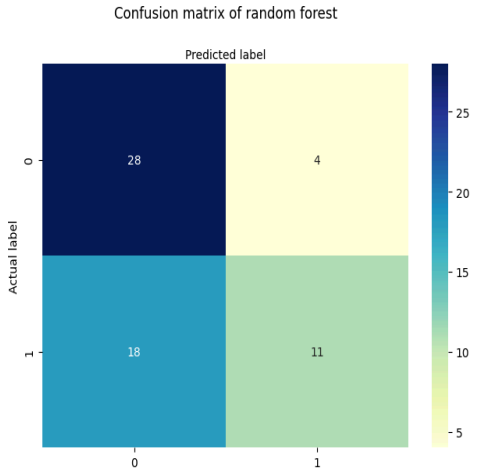
Discussion:
The logistic model didn't do a good job in predicting dead cancer patients, which was the primary reason causing lower accuracy. Since logistic regression is also sensitive to outliers and it requires large sample size when the number of predictor variables is much higher than the number of observations, it might not be an appropriate selection for the data set.

Random Forest

By Zi Li

Dependencies: pandas, numpy, RandomForestClassifier, Sklearn.metrics
Methods: Similar to logistic regression, I chose survival status as the outcome variable, used train_test_split to split the data into 75% training set and 25% testing set and predicted the survival status on the same testing set. The primary advantage of using random forest in binary outcome classification is that it is robust to outliers and overfitting. I further applied cross-validation to find the best hyperparameters such as maximum depth and number of estimators, which can be used to improve model performance. Finally, accuracy and confusion matrix were shown to see how well the predictive performance in each subgroup of the outcome variable.
Results: The model has an accuracy of 63.0%, a maximum depth of 15 and number of estimators of 200.

Figure 2. *Confusion matrix (0 = Alive, 1 = Died):*



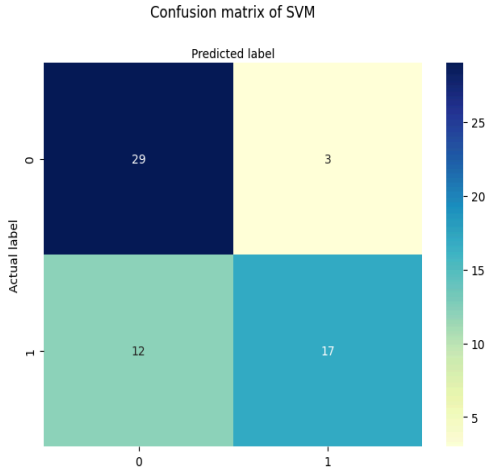
Discussion:
The random forest didn't do a good job in predicting dead cancer patients, which was the primary reason causing lower accuracy. We didn't figure out the primary reasons causing the lower accuracy.

Support Vector Machine (SVM)

By Zi Li

Dependencies: sklearn
Methods: Similar to logistic regression and random forest, I chose survival status as the outcome variable, used train_test_split to split the data into 75% training set and 25% testing set and predicted the survival status on the same testing set. The primary advantage of using support vector machines in binary outcome classification is that it performs well even in high-dimensional spaces, making it suitable for classification tasks where the number of features exceeds the number of samples (particularly for genomics data). Finally, accuracy and confusion matrix were shown to see how well the predictive performance in each subgroup of the outcome variable.
Results: The model has an accuracy of 75.4%.

Figure 3. *Confusion matrix (0 = Alive, 1 = Died):*



Discussion: Compared to random forest and logistic regression, SVM has much higher accuracy, due to its better predictive performance of dead cancer patients. Mathematically speaking, "one reason for SVM's success is its ability to perform extremely complex transformations on input data using kernel functions such as the radial basis function. These transformations enable SVMs to map input data into higher-dimensional feature spaces where linearly inseparable patterns become linearly separable" (1). "By finding an optimal hyperplane that maximally separates different classes in this transformed space, SVMs can effectively classify new instances with minimal error rates" (1).

Neural Network - PyTorch Tabular

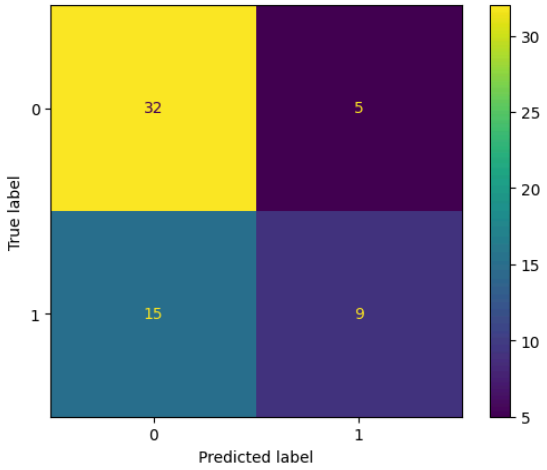
By Soha Shahidi

Dependencies: os, sklearn (metrics & model selection), pytorch_tabular, pandas, numpy
Methods: To build this model, I used a package built off of PyTorch designed specifically for tabular data called pytorch_tabular. The package has 5 basic configuration steps to prepare the data for training. The first is DataConfig, where the target variable, categorical variables, and numeric variables are defined, and transformations are applied. I chose to normalize continuous features to make sure each predictor was contributing evenly. The second is LinearHeadConfig, which I chose to use the defaults, with layers based on the input and output sizes, the activation function being ReLu, dropout being 0.1, and the initialization being kaiming for a nonlinear activation function. ModelConfig allowed me to choose what kind of task the model would do (classification in this case) and which metrics to use (f1 and accuracy, though accuracy was the main decision metric). TrainerConfig contains configurations for how the model would be trained. Since our dataset is small, I decided to shrink the batch size to 16 to allow the model more opportunity to adjust weights for a more accurate prediction. Then, I ran the default OptimizerConfig configurations. The model was fit using the training dataset, which was further split into another training (75%) and validation (25%) set. The model took around 20 seconds to train fully. Predictions were made once the model was trained and compared to the test set held out at the beginning. Results were presented using a confusion matrix, accuracy score, and classification report.
Results: The final model accuracy was 67.2%.

Table 4. *Classification report:*

	precision	recall	f1-score	support
Alive	0.68	0.86	0.76	37
Died	0.64	0.38	0.47	24
accuracy			0.67	61
macro avg	0.66	0.62	0.62	61
weighted avg	0.67	0.67	0.65	61

Figure 4. *Confusion matrix (0 = Alive, 1 = Died):*



Discussion: This model performed fairly poorly, which makes sense since there were so few subjects to build the model off of. Neural networks work best when given a lot of data to work off of, especially in cases where data is high-dimensional as it is here. This manifested as the learning rate finder diverging early (usually around 82%), and the model only running 10 epochs out of the 100 epoch limit.

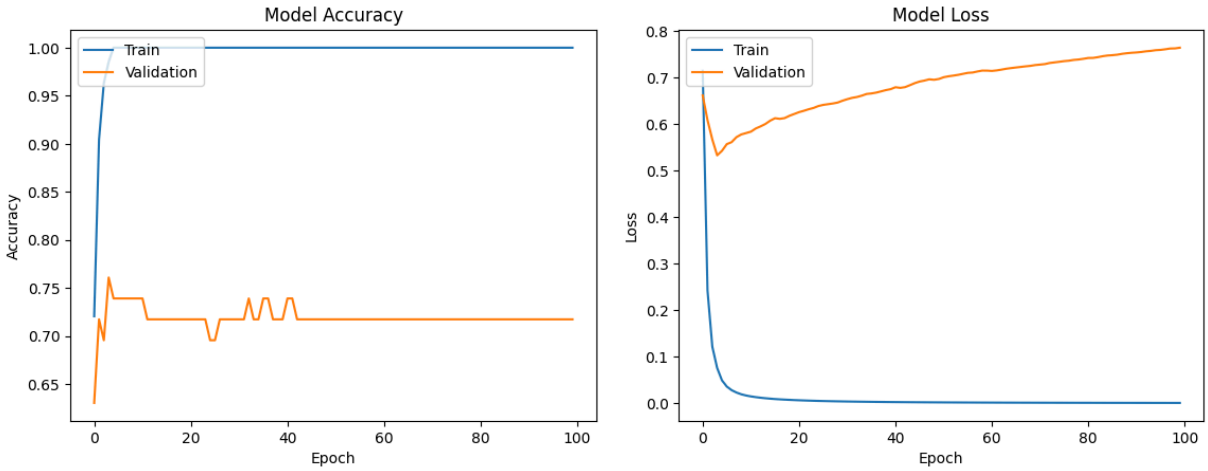
I had many challenges when building this model due to the lack of data, and I had to figure out the best way to tune the model to work with this limitation. The step I found worked the best was increasing the number of epochs (originally I had it at 4-5 epochs) while also decreasing the batch size, allowing the model to adjust itself to the data that was there more effectively. In a larger dataset, having such small batches would be a computational nightmare, but in this case, the dataset was small enough that small batches were feasible. The model also had an issue with its F1 score, as every time I ran the model it was exactly the same as the accuracy. This was determined to be some kind of bug with the model itself, as calculating the F1 score manually or with the classification report table (Table 1) gave a number that made more sense. All in all, this model showed that neural networks work best when given enough data to work with.

Neural Network - Keras

By Soha Shahidi

Dependencies: sklearn (metrics, model selection, & preprocessing), keras, pandas, numpy, matplotlib
Methods: To write the Keras model, I followed a tutorial, which is linked below at (1). I started by coding the only categorical predictor, gender, and the outcome, survival status, into dummy variables. Then, I split the data into a training and test set as described above. Data was standardized using a standard scalar. The model itself contained 5 layers: an input later the size of the training set, 3 ReLu dense layers that went from 128 nodes to 64 to 32, and an output layer with a sigmoid activation function for a binary outcome. The model was compiled with binary cross entropy, an Adam optimizer with a learning rate of 0.0001, and accuracy as a decision metric. The model was originally run with 100 epochs, and accuracy and loss were plotted for the training vs. validation sets. This model took 3.8 seconds to run.

Figure 5. *Accuracy and Loss plots for Training and Validation sets.*

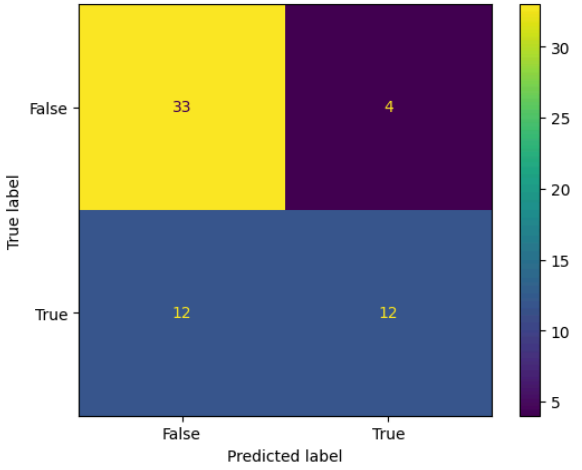


I chose 15 epochs as a sweet spot between accuracy and loss, and ran the model again on the full training set. This model took less than a second to run. I then ran predictions and compared them to the training set. Results were presented using an accuracy score, classification report, and confusion matrix.
Results: The model had a 73.8% accuracy.

Table 5. *Classification report:*

	precision	recall	f1-score	support
Alive	0.73	0.89	0.80	37
Died	0.75	0.50	0.60	24
accuracy			0.74	61
macro avg	0.74	0.70	0.70	61
weighted avg	0.74	0.74	0.72	61

Figure 6. *Confusion Matrix (0 = Alive, 1 = Died):*



Discussion: Since the PyTorch neural network had issues, I decided to try using another neural network package, Keras, which is known to be easier to work with. This ended up being true. The model performed better as well, with a 6.6% higher accuracy. This still is not the best, but certainly higher than the PyTorch model. However, looking at Figure 5, we can see that there is a major discrepancy between the training and validation datasets in both accuracy and loss. This is likely due to the small sample size being used.

The biggest limitation of this model, besides the small sample size, is that this model is fairly unstable. Each time it is run, it randomizes again, and the accuracy score and confusion matrix change significantly. This instability is also likely due to the small sample size, with a small change in which subjects are being trained on affecting the entire model greatly.

XGBoost

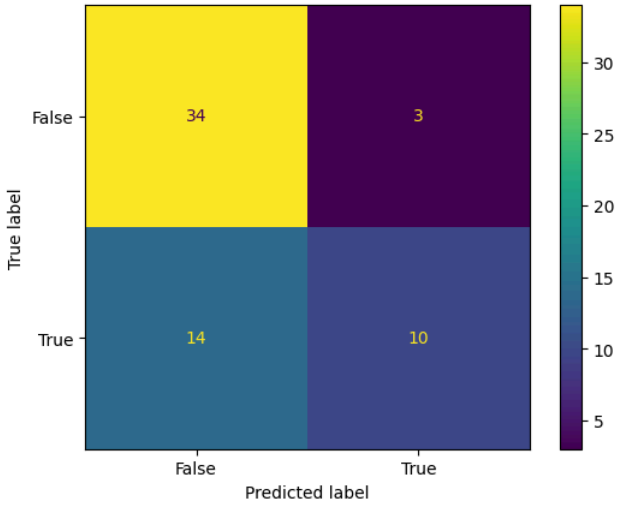
By Soha Shahidi

Dependencies: sklearn (metrics & model selection), xgboost, pandas, numpy, matplotlib
Methods: To write the xgboost section, I mostly followed the same steps as our exercise 4 and 5. I tuned the model using a hyperparameter grid and 5-fold cross validation, with the tuning grid containing 50, 100, 200, or 500 estimators, a max tree depth of 1, 2, 3, 4, 5, or 6, and an eta of 0.1, or 0.3. The decision metric was accuracy. The best model had a cross validation accuracy score of 75.8%, and the hyperparameters chosen were a max depth of 5, 200 estimators, and an eta of 0.1. This process took over 13 minutes. I trained another xgboost model with the winning hyperparameters on the full training set, which took 8 seconds. The model's predictions were compared to the test set, and an accuracy score, classification report, and confusion matrix were generated.
Results: The accuracy for this model was 72.1%.

Table 6. *Classification Report:*

	precision	recall	f1-score	support
Alive	0.71	0.92	0.80	37
Died	0.77	0.42	0.54	24
accuracy			0.72	61
macro avg	0.74	0.67	0.67	61
weighted avg	0.73	0.72	0.70	61

Figure 7. *Confusion matrix (0 = Alive, 1 = Died):*



Discussion: The final model I chose to try was xgboost, as it is the gold standard among boosted models and should perform well with high dimensional data. This model performed fairly poorly overall, though compared to the other models here, it performed fairly well. Again, this seems to be a limitation of the small dataset. This model did have the highest number of true negatives out of every model, implying a good specificity.

Challenges and Limitations

The first challenge to overcome was the number of columns in the dataset. The original plan for our project was to do feature selection to select only the most important gene expression columns for predicting survival status, but we found that each column on its own contributed very little to determining survival (~0.003). The models ran well enough using all gene expression columns, so we decided to skip the feature selection step and go straight to testing out the models.

The second challenge was the number of rows. This dataset has very few subjects, which is not ideal for prediction problems, especially using models that need a lot of data like neural nets. This proved to be a significant limitation on the accuracy of our predictions, with most models scoring between 60-70%.

Results

Among all the binary outcome classification models we tried, SVM, the Keras neural network, and xgboost all have accuracy higher than 70%. The SVM has the best predictive performance in the high dimensional data we have, with an accuracy of 75.4%. The random forest has the worst predictive performance, with an accuracy score of 63.0%.

Conclusion

In conclusion, we see that for the KIRC dataset, SVM models perform the best. This is likely due to the way they construct hyperplanes with high dimensional data, rather than relying on the sheer magnitude of data to assign weights like a neural network might. Despite this better performance, the SVM still performed mediocrely overall, if a good performance is thought to be above 80% accuracy. This project shows that small datasets with few subjects are difficult to model, even when (or especially when) there are plenty of predictors to work with.

Sources

(1). https://www.redshiftrecruiting.com/support-vector-machines-high-dimensional-spaces - the power of support vector machines: effectiveness in higher dimensional spaces
(2). https://www.pythonicly.com/tensorflow-keras-binary-classification-on-tabular-data/ - Keras tutorial for tabular data
(3). https://not.badmath.org/ - Course website